The AArch64 processor (aka arm64), part 12: Memory access and alignment

devblogs.microsoft.com/oldnewthing/20220810-00

August 10, 2022

Raymond Chen

Accessing memory is done primarily through load and store instructions.

```
; load word or doubleword register
ldr     Rn/zr, [...]

; load unsigned byte
ldrb    Wn/zr, [...]

; load signed byte
ldrsb   Rn/zr, [...]

; load unsigned halfword
ldrh    Wn/zr, [...]

; load signed halfword
ldrsh   Rn/zr, [...]

; load signed word
ldrsw   Xn/zr, [...]

; load pair of registers
ldp     Rd1/zr, Rd2/zr, [...]

; load pair of registers as signed word
ldpsw   Xd1/zr, Xd2/zr, [...]
```

AArch64 does not have AArch32's `LDM` instruction for loading up to 13 registers at once. As a consolation present, it gives you a `LDP` instruction for loading two registers, either 32-bit or 64-bit, from consecutive bytes of memory. (The first register uses the lower address.) The `LDP` instruction is commonly used with the 64-bit registers to load spilled registers from the stack.

There is a corresponding selection of instructions for storing to memory, but obviously the sign extension variations are not relevant.

```
; store word or doubleword register
str     Rn/zr, [...]

; store byte
strb    Wn/zr, [...]

; store halfword
strh    Wn/zr, [...]

; store pair of registers
stp     Rd1/zr, Rd2/zr, [...]
```

Not all addressing modes are available for all variations. This is not something you worry about when reading assembly language, but it's something you need to keep in mind when writing it.

Size	`[Xn/sp, #imm]` (−256 … +255)	`[Xn/sp, #imm]` `[Xn/sp, #imm]!` `[Xn/sp], #imm`	`[pc, #imm]` (±1MB)	`[Xn/sp, Rn/zr, extend]`
byte	•	•		•
halfword	•	•		•
word	•	•	loads only	•
doubleword	•	•	loads only	•
pair		•		

The reach of the second column is is (0 … 4095) × *size*, except that the reach of the the register pairs is (−64 … 63) × *size*.

All operand sizes support register indirect with offset. Only word and doubleword support *pc*-relative (and even those are supported only for loads). And register pairs support only register indirect with offset.

There are some ambiguous encodings, because a constant offset in the range 0 … 255 that is a multiple of the operand size can be encoded either as a 9-bit signed byte offset, or as a 12-bit unsigned element offset. By default, assemblers will use the 12-bit unsigned element offset, but you can force the 9-bit signed byte offset by changing the opcode from `LDxxx` and `STxxx` to `LDUxxx` and `STUxxx`. The `U` stands for *unscaled*.

Windows enables automatic unaligned access fixups. Simple unaligned memory accesses are fixed up automatically by the processor, but you lose atomicity: It is possible for an unaligned memory access to read a torn value. Any such tearing is at the byte level.

Original value	`12`	`34`	`56`	`78`	aligned
Processor 1 reads					misaligned
Processor 2 writes	`AB`	`CD`	`EF`	`01`	aligned

The misaligned halfword read from processor 1 could produce `34|56`, `34|EF`, `CD|56`, or `CD|EF`. But it won't produce `3D|EF`.

You can still take alignment faults if the misaligned memory access is fancy, such as a locked load, store exclusive, or a load with a memory barrier. We'll learn about these special memory accesses next time.

Raymond Chen

Follow